A NEW APPROACH TO RECURSIVE PROGRAMS

Zohar Manna, et al

Stanford University

Prepared for:

Office of Naval Research

December 1975

Stanford Artificial Intelligence Laboratory
Memo AIM-276

December 1975

Computer Science Department
Report No. STAN-CS-75-539

A NEW APPROACH TO RECURSIVE PROGRAMS

by

ZOHAR MANNA* AND ADI SHAMIR**

Research sponsored by

Advanced Research Projects Agency
ARPA Order No. 2494

COMPUTER SCIENCE DEPARTMENT
Stanford University

Stanford Artificial Intelligence Laboratory
Memo AIM-276

December 1975

Computer Science Department
Report No. STAN-CS-75-539

A NEW APPROACH TO RECURSIVE PROGRAMS

by

ZOHAR MANNA* AND ADI SHAMIR**

ABSTRACT

In this paper we critically evaluate the classical least-fixedpoint approach towards recursive programs. We suggest a new approach which extracts the maximal amount of valuable information embedded in the programs. The presentation is informal, with emphasis on examples.

*Formerly of the Weizmann Institute of Science, Rehovot, Israel Present Address: Artificial Intelligence Laboratory, Stanford University, Stanford, California 94305 **Formerly of the Weizmann Institute of Science, Rehovot, Israel Present Address: Computer Science Department, University of Warwick, Warwick, England

This research was supported by the Advanced Research Project: Agency of the Department of Defense under Contract DAHC 15-73-C-0435 . The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, ARPA, or the U. S. Government.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER STAN-CS-75-539, AIM276	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A New Approach to Recursive Programs		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Zohar Manna and Adi Shamir		8. CONTRACT OR GRANT NUMBER(s) DAHC15-73-C-0435
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory Stanford University Stanford, California 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order 2494
11. CONTROLLING OFFICE NAME AND ADDRESS Col.Dave Russell, Dep. Dir., ARPA, IPT, ARPA Headquarters, 1400 Wilson Blvd. Arlington, Virginia 22209		12. REPORT DATE December 1975
		13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Philip Surra, ONR Representative Durand Aeronautics Building Room 165 Stanford University Stanford, California 94305		15. SECURITY CLASS. (of this report) 29
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Releasable without limitations on dissemination.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) In this paper we critically evaluate the classical least-fixed point approach towards recursive programs. We suggest a new approach which extracts the maximal amount of valuable information embedded in the programs. The presentation is informal, with emphasis on examples.		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

I. Introduction

The classical stack implementation of recursive programs does not always give results that correspond to our naive intuitive expectations. For instance, one might expect the program

$$F(x) \Leftarrow 0 \cdot F(x)$$

over, say, the natural numbers, to be identical to the program

$$F(x) \Leftarrow 0,$$

since for any number y, $0 \cdot y = y$. Similarly, the program

$$F(x) \Leftarrow \textit{if } F(x)=0 \textit{ then } 0 \textit{ else } 0$$

would be expected to yield the zero function. Since the test $F(x)=0$ is irrelevant, nothing but 0 can be produced as an output. However, stack implementations and the conventional theory of programs dictate that both of these programs be undefined for all inputs. Users of recursion are so accustomed to this implementation that they are no longer surprised at this unintuitive interpretation, and never stop to consider any alternative meanings of recursive programs.

A recursive program, such as those above, looks like an implicit functional equation relating the values of the function variable F. Such an equation may in general have many possible solution functions (*fixedpoints*). Since there is no unique solution, the semantics of recursive programs is selected rather than implied. The classical stack implementation yields one solution, the ***least defined fixedpoint*** of the program. As we have seen above, the blind selection of the least defined solution is inadequate, because a recursive program often contains more information than this solution exhibits.

In this paper we suggest the selection of a different and more defined solution, which always exists and which contains as much information as possible.

In Section II we discuss various possible approaches towards recursive programs, in an attempt to characterize the "best" one. On the basis of this discussion we introduce our new *optimal fixedpoint* approach in Section III, which is exemplified in Section IV. Various techniques for proving properties of optimal fixedpoints are presented in Section V.

This paper is an informal exposition of the optimal fixedpoint theory. More formal treatment is given in Manna and Shamir [1975] and Shamir [1976].

II. Recursive Programs and Their Fixedpoints

Consider, as a typical example, the following recursive program P1 over the natural numbers [see note (i)]:

P1: $F(x,y) \Leftarrow$ if $x=0$ then y else $F(F(x,y-1),F(x-1,y))$.

Any solution function to this program must satisfy the relations dictated by the program, i.e.,

(a) $F(0,y) = y$ for all y, and
(b) $F(x,y) = F(F(x,y-1), F(x-1,y))$ for all $x \neq 0$ and all y.

Let us analyze what functions satisfy these two conditions.

The main part of this program is the functional

$\tau[F]$: *if* $x=0$ *then* y *else* $F(F(x,y-1),F(x-1,y))$,

in which the symbol F is considered as a *function variable*. Given any partial function $f(x,y)$, the result of substituting f for F yields a new partial function, denoted by $\tau[f]$. For example, if we substitute the function

$f(x,y) = y$

for $F(x,y)$, we obtain the function

$$\begin{aligned}\tau[f](x,y) &= \textit{if } x=0 \textit{ then } y \textit{ else } f(f(x,y-1), f(x-1,y)) \\ &= \textit{if } x=0 \textit{ then } y \textit{ else } f(y-1,y) \\ &= y.\end{aligned}$$

Thus, the function $f(x,y)$ has the interesting property that $f(x,y)=\tau[f](x,y)$, that is, f is a solution function to the functional equation $F(x,y)=\tau[F](x,y)$. Since f does not change under the application of τ , it is said to be a *fixedpoint* of the given recursive program.

An entirely different function which is a fixedpoint of the program is:

$g(x,y) = max(x,y)$.

Substituting g for F in $\tau[F]$, we obtain:

$$\tau[g](x,y) = \textit{if } x=0 \textit{ then } y \textit{ else } \mathit{max}(\mathit{max}(x,y-1),\mathit{max}(x-1,y)) \ .$$

By the definition of *max*, this can be simplified to:

$$\begin{aligned}\tau[g](x,y) &= \textit{if } x=0 \textit{ then } y \textit{ else } \mathit{max}(x,y-1,x-1,y)\\ &= \textit{if } x=0 \textit{ then } \mathit{max}(x,y) \textit{ else } \mathit{max}(x,y)\\ &= \mathit{max}(x,y) \ .\end{aligned}$$

Thus $g(x,y)$ is a fixedpoint of the recursive program P1.

Yet another example of a fixedpoint is the partial function:

$$l(x,y) = \textit{if } x=0 \textit{ then } y \textit{ else undefined} \ .$$

To show that this function is indeed a fixedpoint of our recursive program, we substitute l in τ, treating *undefined* as any other value. For this purpose we make the general assumption that all functions and predicates appearing in τ are "naturally extended," in the sense that they are *undefined* whenever at least one of their arguments is *undefined*. Thus, we have:

$$\begin{aligned}\tau[l](x,y) &= \textit{if } x=0 \textit{ then } y \textit{ else } l(l(x,y-1),l(x-1,y))\\ &= \textit{if } x=0 \textit{ then } y \textit{ else}\\ &\quad\ l(\textit{if } x=0 \textit{ then } y-1 \textit{ else undefined},\ l(x-1,y))\\ &= \textit{if } x=0 \textit{ then } y \textit{ else } l(\textit{undefined},\ l(x-1,y))\\ &= \textit{if } x=0 \textit{ then } y \textit{ else}\\ &\quad\ \textit{if undefined}=0 \textit{ then } l(x-1,y) \textit{ else undefined}\\ &= \textit{if } x=0 \textit{ then } y \textit{ else undefined} \ .\end{aligned}$$

These three functions do not exhaust the set of all fixedpoints of the program. An example of an infinite class of fixedpoints (indexed by the function a over the natural numbers) is:

$$h_a(x,y) = \textit{if } x=0 \textit{ then } y \textit{ else } a(x).$$

A function $h_a(x,y)$ can be shown to be a fixedpoint of the program, provided that the function $a(n)$ satisfies:

$$a(n)\neq 0 \text{ and } a(a(n))=a(n) \quad \text{for all} \quad n > 0.$$

Examples of functions satisfying this condition are the identity function, any non-zero constant function, or the function which assigns to any natural number n its greatest prime factor.

There are actually infinitely many more distinct fixedpoints, the exact characterization of which is quite complicated. We can thus see that the set of all fixedpoints of the program may contain many functions with extremely diversified behavior. All these functions can be considered as "solutions" to the equation represented by our recursive program.

Some of these fixedpoints are related by the "less defined or equal" relation. We say that a function r(x,y) is *less defined or equal* to s(x,y), or that s(x,y) is *more defined or equal* to r(x,y), if for any pair of natural numbers (a,b) , if r(a,b) is defined than s(a,b) is also defined and has the same value; thus, either r(a,b) is undefined or else r(a,b)=s(a,b) . Note that a function r(x,y) may be neither "less defined or equal" nor "more defined or equal" to s(x,y).

This relation introduces some structure into the set of all fixedpoints of a recursive program. A fixedpoint is called *least (defined)* if it is less defined or equal to any other fixedpoint of the program. Dually, a fixedpoint is called *greatest (defined)* if it is more defined or equal to any other fixedpoint.

Among the fixedpoints of the program P1 , the fixedpoint:

$$l(x,y) = \textit{if } x=0 \textit{ then } y \textit{ else undefined}$$

stands out. Since any fixedpoint of P1 must be defined as y for x=0, it is clearly the program's least fixedpoint.

Least fixedpoints of recursive programs have long attracted the attention of computer science theoreticians for three main reasons (see, e.g., Manna [1974]):

(a) Any recursive program must have a (unique) least fixedpoint. Thus the least fixedpoint can be used to unambiguously define the "meaning" of recursive programs.
(b) The classical stack implementation of recursive programs computes the least fixedpoint of the program.
(c) There are powerful methods for proving properties of the least fixedpoint of programs.

As a result, the least fixedpoint was chosen as the "proper" solution of recursive programs and other fixedpoints were absolutely discarded by researchers from further consideration. However, we have an important objection to this choice: it contradicts the intuitive concept that the more defined the solution, the more valuable it is. Indeed, there are many recursive programs for which the least fixedpoint does not contain all the useful information embedded in the program, information which is contained in more defined fixedpoints.

Consider, for example, the following recursive program P2 for solving the discrete form of the Laplace equation, where $F(x,y)$ maps pairs of integers in $[0,100] \times [0,100]$ into reals:

```
P2:  F(x,y) = if x=0 then 2y
              else if x=100 then 3y+300
              else if y=0 then 3x
              else if y=100 then 4x+200
              else [F(x-1,y)+F(x,y-1)+F(x+1,y)+F(x,y+1)]/4 .
```

This recursive program has exactly two fixedpoints:

$$f(x,y) = \begin{cases} 2y & \text{if } x=0 \\ 3y+300 & \text{if } x=100 \\ 3x & \text{if } y=0 \\ 4x+200 & \text{if } y=100 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

and

$$g(x,y) = 3x+2y+(x \cdot y)/100 \quad \text{for} \quad 0 \le x,y \le 100 \quad .$$

There is no doubt that the second (totally defined) fixedpoint $g(x,y)$ contains much more valuable information than the (mostly undefined) function $f(x,y)$. Moreover, it is quite obvious that any programmer writing such a recursive program unconsciously thinks about the function $g(x,y)$ as the "solution" of the functional equation represented by the program. Thus, the arbitrary selection of the least fixedpoint as the "proper solution" seems a poor choice in this case.

This example might suggest a turn to the other extreme - considering greatest fixedpoints rather than least fixedpoints. Unfortunately, there are many programs for which there is no such greatest fixedpoint, as program P1 shows: There is no function which is more defined than all the fixedpoints exhibited.

A more modest approach could be the selection of a *maximal fixedpoint*, i.e., a fixedpoint which is not less defined than any other fixedpoint. However, there are difficulties with this choice too. While any recursive program has such a fixedpoint, it may have more than one. This is demonstrated by program P1, in which the functions $f(x,y)$, $g(x,y)$ and $h_a(x,y)$ are all examples of total, and therefore maximal, fixedpoints of P1. This indicates that P1 is an "underdefined" recursive program - the relations stated between values of F for various arguments (x,y) are not sufficient to uniquely determine one defined value of the fixedpoint. Thus a randomly chosen maximal fixedpoint is by no means superior to the least fixedpoint $f(x,y)$ in this case.

An artificial example which illustrates this problem is:

P3: F(x) <= F(x)

over, say, the set of the natural numbers. Any partial function over the natural numbers is clearly a fixedpoint of this extremely "underdefined" program. The least fixedpoint of P3 is the totally undefined function, and every total function over the natural numbers is a maximal fixedpoint. In such a case, the least fixedpoint seems the most appropriate solution, since no other fixedpoint can be considered a more "valuable" solution of this program.

III. The Optimal Fixedpoint

Thus far we have objected to the classical least fixedpoint and the proposed greatest and maximal fixedpoint approaches to recursive programs. We now suggest a new approach -- the "optimal fixedpoint approach". It combines the nice properties of all the above approaches in that the fixedpoint selected always uniquely exists, and it supplies the maximal amount of valuable information embedded in the program. Thus in the three examples considered so far, the new approach will select the least fixedpoint in the "underdefined" programs P1 and P3, but will select the desired total fixedpoint (which differs from the least fixedpoint) in the Laplace program P2.

In order to develop the new approach we first introduce the notion of "consistency". Two functions are said to be *consistent* if they have identical values for any argument for which both are defined. For example, let

$$f_1(x) = \begin{cases} 0 & \text{if } x=0 \\ \textit{undefined} & \text{if } x=1 \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x) = \begin{cases} 0 & \text{if } x=0 \\ 1 & \text{if } x=1 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

$$f_3(x) = \begin{cases} 0 & \text{if } x=0 \\ 2 & \text{if } x=1 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

Then f_1 and f_2 are consistent, as are f_1 and f_3. However, f_2 and f_3 are not consistent, since for $x=1$ both are defined and have different values. Note that no two of these functions are related by the "less defined or equal" relation.

Two consistent functions can be regarded as being "approximately the same": One function may be defined for several arguments at which the other is undefined, and vice versa; but the two functions cannot have contradictory defined values. They can be considered as two incomplete representations of the same knowledge, and one can define a function which is more defined than both of them, thus being superior to both partial representations.

We now define a fixedpoint f of a program P to be *fxp-consistent* if f is consistent with any other fixedpoint g of P . That is, whenever f is defined, say f(x)=a , then for any other fixedpoint g , either g(x) is *undefined* or g(x)=a . Thus the value a is implicitly defined by the program as the only possible defined solution at x. Every recursive program has at least one fxp-consistent fixedpoint, since the least fixedpoint of the program is less defined than (and thus consistent with) any other fixedpoint of the program. Thus, the classical least fixedpoint is one of these valuable fixedpoints, but only one of many.

The fxp-consistent fixedpoints can be considered as the only genuine solutions of a recursive program, since only they contain uniquely determined values. We can thus concentrate our attention on the subset of fxp-consistent fixedpoints rather than on the set of all fixedpoints of the program. In this restricted set of solutions we are naturally interested in maximally defined solutions of the program. While the greatest fixedpoint approach was not applicable to the set of all fixedpoints of the program, we now fortunately have [see note (ii)]:

Basic Theorem: The set of all fxp-consistent fixedpoints always contains a (unique) greatest element.

Let us now look at the set of fixedpoints from a different point of view. Previously, we discussed the possibility of selecting a maximal fixedpoint as the "proper" solution of the program. This approach was not applicable, since the program may have infinitely many such solutions with no information common to all of them, and no one of which seems superior to the others. A natural way to resolve this problem is to find a fixedpoint which extracts the unanimity among these maximal fixedpoints, thus being a satisfactory representative of all of them. Such a fixedpoint can be obtained by considering the fixedpoints which are less defined than all the maximal fixedpoints. For these fixedpoints we again have [see note (iii)]:

Basic Theorem: The set of fixedpoints which are less defined than all maximal fixedpoints of the program, has a (unique) greatest element.

We have thus arrived at two possible definitions of the "most desired solution" of a recursive program, the first by ascending as much as possible from the least fixedpoint in the set of fxp-consistent fixedpoints, and the second by descending from the maximal fixedpoints.

It is quite natural to relate these two "desired solutions" of a recursive program. Surprisingly enough, these two fixedpoints always coincide, and we call the fixedpoint thus defined the *optimal fixedpoint* of the program.

By the definition of the optimal fixedpoint, it follows that any recursive program has a unique optimal fixedpoint. If the program has only one fixedpoint which is fxp-consistent, the optimal fixedpoint coincides with the classical least fixedpoint. On the other hand, if the program has a unique maximal fixedpoint, the optimal fixedpoint coincides with it. In all other cases, the optimal fixedpoint "floats" somewhere in the set of all fixedpoints. We illustrate this with the following diagram (see Fig. 1), which summarizes some of the structural properties of the set of fixedpoints of recursive programs. In this diagram an upper section (Fig. 2A) represents the set of all fixedpoints which are more defined or equal to f ; similarly, a lower section (Fig. 2B) represents the set of all fixedpoints which are less defined or equal to f . The "strategic position" of the optimal fixedpoint is clearly visible.

IV. A Detailed Example

Consider the following family of recursive programs over the natural numbers:

$$P_{i,j}: \quad F(x) \Leftarrow \textit{if } x=0 \textit{ then } i \textit{ else } j \cdot F(F(x-1)) .$$

We shall investigate the structure of the set of fixedpoints for a few recursive programs in this family, thus illustrating the behavior of the optimal fixedpoint approach in various situations. In order to systematically analyze the possible values of fixedpoints for some x=a , we evaluate the term F(a) by repeatedly substituting $\tau[F]$ for various occurrences of F . Note that we make use of the fact that F represents a fixedpoint of the program, but not necessarily the least fixedpoint or the optimal fixedpoint.

Programs $P_{0,j}$:

$$F(x) \Leftarrow \textit{if } x=0 \textit{ then } 0 \textit{ else } j \cdot F(F(x-1)) .$$

Let us analyze the possible values of F for successive arguments x :

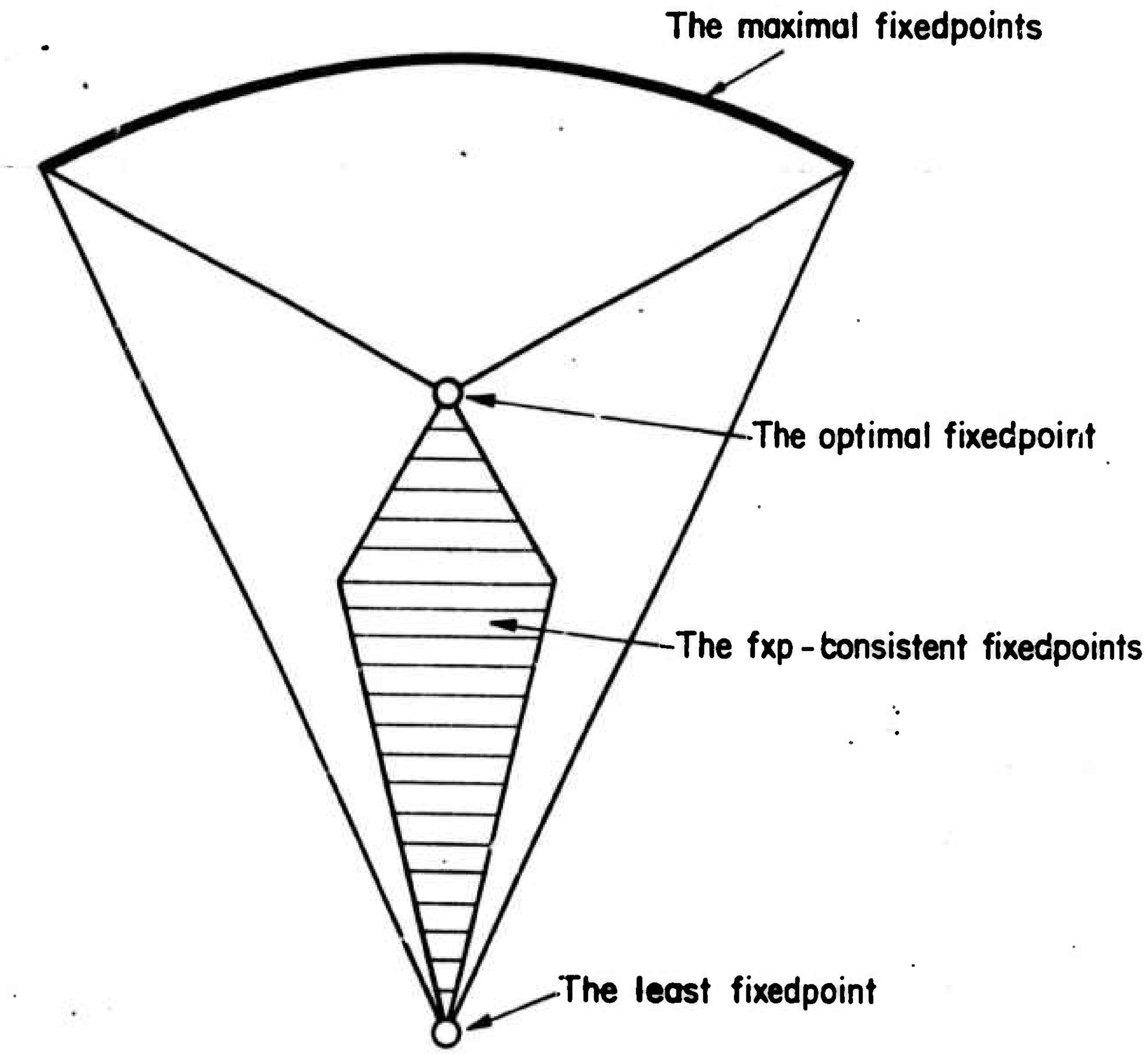

Fig. 1. The fixedpoints of a recursive program .

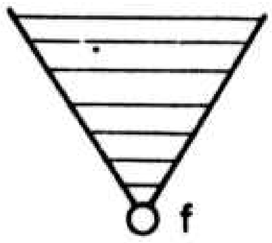

Fig. 2A

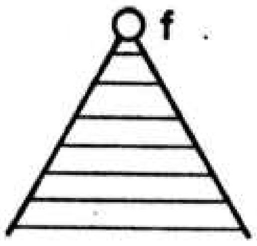

Fig. 2B

$$F(0) = \textit{if } 0=0 \textit{ then } 0 \textit{ else } j\cdot F(F(0-1)) = 0$$

$$\begin{aligned} F(1) &= \textit{if } 1=0 \textit{ then } 0 \textit{ else } j\cdot F(F(1-1)) \\ &= j\cdot F(F(0)) = j\cdot F(0) = j\cdot 0 = 0 \end{aligned}$$

$$\begin{aligned} F(2) &= \textit{if } 2=0 \textit{ then } 0 \textit{ else } j\cdot F(F(2-1)) \\ &= j\cdot F(F(1)) = j\cdot F(0) = j\cdot 0 = 0 \end{aligned}$$

It can be easily shown (by induction) that $F(x)=0$ for any natural number x . Thus for any j , the program $P_{0,j}$ has exactly one fixedpoint:

$$f(x) = 0 \quad \text{for any natural number } x \text{ .}$$

It is clearly the program's least fixedpoint as well as the program'e optimal fixedpoint.

The behavior of the programs changes drastically when we take i to be 1 rather than 0 .

Program $P_{1,0}$:

$$F(x) \Leftarrow \textit{if } x=0 \textit{ then } 1 \textit{ else } 0\cdot F(F(x-1)) \text{ .}$$

The value of $F(0)$ is clearly 1 , by a direct application of the recursive definition. For $x=1$, however, we get:

$$\begin{aligned} F(1) &= \textit{if } 1=0 \textit{ then } 1 \textit{ else } 0\cdot F(F(1-1)) \\ &= 0\cdot F(F(0)) = 0\cdot F(1) \text{ .} \end{aligned}$$

We now have exactly two possible values for $F(1)$:

$$F(1) = \textit{undefined} \quad \text{or} \quad F(1) = 0 \text{ .}$$

Selecting the first possibility, $F(1) = \textit{undefined}$, we obtain:

$$\begin{aligned} F(2) &= \textit{if } 2=0 \textit{ then } 1 \textit{ else } 0\cdot F(F(2-1)) \\ &= 0\cdot F(F(1)) = 0\cdot F(\textit{undefined}) \\ &= 0\cdot(\textit{if undefined}=0 \textit{ then } 1 \textit{ else } 0\cdot F(F(\textit{undefined}-1))) \\ &= 0\cdot \textit{undefined} = \textit{undefined} \text{ .} \end{aligned}$$

Continuing in this way, we get the fixedpoint:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise .} \end{cases}$$

However, if we select the second possibility, $F(1)=0$, we have to continue in the following way:

$$F(2) = \textit{if } 2=0 \textit{ then } 1 \textit{ else } 0 \cdot F(F(2-1))$$
$$= 0 \cdot F(F(1)) = 0 \cdot F(0) = 0 \cdot 1 = 0,$$

and so on. We thus get the fixedpoint:

$$g(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{otherwise .} \end{cases}$$

The functions $f(x)$ and $g(x)$ are clearly the only possible fixedpoints of the program. Since $f(x)$ is less defined than $g(x)$, $f(x)$ is the program's least fixedpoint while $g(x)$ is the program's optimal fixedpoint.

Program $P_{1,1}$:

$$F(x) \Leftarrow \textit{if } x=0 \textit{ then } 1 \textit{ else } F(F(x-1)) \text{ .}$$

The value of $F(0)$ is necessarily 1 . Evaluating $F(1)$, we get:

$$F(1) = \textit{if } 1=0 \textit{ then } 1 \textit{ else } F(F(1-1))$$
$$= F(F(0)) = F(1) \text{ ,}$$

and thus any natural number (as well as the value *undefined*) is a solution of this equation. If we choose $F(1)=$*undefined* , we get (exactly as in program $P_{1,0}$) the fixedpoint:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise .} \end{cases}$$

Since any other fixedpoint of $P_{1,1}$ must also be 1 for $x=0$, $f(x)$ is clearly the program's least fixedpoint.

Suppose we choose $F(1) = 0$. We then continue with:

$$\begin{aligned} F(2) &= \textit{if } 2=0 \textit{ then } 1 \textit{ else } F(F(2-1)) \\ &= F(F(1)) = F(0) = 1 \end{aligned}$$

$$\begin{aligned} F(3) &= \textit{if } 3=0 \textit{ then } 1 \textit{ else } F(F(3-1)) \\ &= F(F(2)) = F(1) = 0 \end{aligned}$$

and so on. We thus get the fixedpoint:

$$g(x) = \begin{cases} 1 & \text{if } x \text{ is even} \\ 0 & \text{if } x \text{ is odd .} \end{cases}$$

If we take $F(1)=1$, we obtain:

$$\begin{aligned} F(2) &= \text{if } 2=0 \textit{ then } 1 \textit{ else } F(F(2-1)) \\ &= F(F(1)) = F(1) = 1 \ , \end{aligned}$$

and so on. We thus obtain the fixedpoint:

$$h(x) = 1 \quad \text{for any natural number } x \ .$$

If we take $F(1) = 2$, we get:

$$\begin{aligned} F(2) &= \text{if } 2=0 \textit{ then } 1 \textit{ else } F(F(2-1)) \\ &= F(F(1)) = F(2) \end{aligned}$$

and again we may choose any desired value for $F(2)$ (including the value *undefined*).

It is possible to continue this detailed analysis and find infinitely many more fixedpoints of $P_{1,1}$. But in order to characterize the optimal fixedpoint of this program it suffices to consider just one more fixedpoint:

$$k(x) = x+1 \quad \text{for any natural number } x \ .$$

Since the optimal fixedpoint should be less defined than both maximal fixedpoints $h(x)$ and $k(x)$, it cannot be defined for any $x>0$ (for any such x both $h(x)$ and $k(x)$ are defined and $h(x) \neq k(x)$) . Therefore the program's optimal fixedpoint coincides in this case with the program's least fixedpoint $f(x)$.

Program $P_{1,2}$:

$$F(x) \Leftarrow \textit{if } x=0 \textit{ then } 1 \textit{ else } 2 \cdot F(F(x-1)) \ .$$

As before, all fixedpoints of $P_{1,2}$ are defined as 1 for $x=0$. For $x=1$ we have

$$F(1) = \textit{if } 1=0 \textit{ then } 1 \textit{ else } 2\cdot F(F(0)) = 2\cdot F(1) .$$

We have arrived at an equation (for the value of $F(1)$) which has exactly two solutions:

$$F(1) = \textit{undefined} \quad \text{or} \quad F(1) = 0 .$$

If we decide to take the value $F(1) = \textit{undefined}$, we again get the fixedpoint:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

which is the program's least fixedpoint.

Choosing the other possibility, i.e., $F(1)=0$, we get:

$$F(2) = 2\cdot F(F(1)) = 2\cdot F(0) = 2 ,$$

$$F(3) = 2\cdot F(F(2)) = 2\cdot F(2) = 4 ,$$

and finally:

$$F(4) = 2\cdot F(F(3)) = 2\cdot F(4) .$$

The values for $F(2)$ and $F(3)$ were implied, once we chose $F(1)=0$. But for $F(4)$, we again have to choose between the two possible solutions of the equation, namely,

$$F(4) = \textit{undefined} \quad \text{or} \quad F(4)=0 .$$

If we choose $F(4)=\textit{undefined}$, then an argument similar to the one used previously shows that for any $x>4$, $F(x)=\textit{undefined}$. Thus we have the fixedpoint

$$g(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1 \\ 2 & \text{if } x=2 \\ 4 & \text{if } x=3 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

However, if we choose $F(4)=0$, we must continue as follows

$$F(5) = 2\cdot F(F(4)) = 2\cdot F(0) = 2$$

$$F(6) = 2\cdot F(F(5)) = 2\cdot F(2) = 4$$

$$F(7) = 2\cdot F(F(6)) = 2\cdot F(4) = 0,$$

and so on. The periodic function thus obtained is defined for any natural number x as:

$$h(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1+3i \\ 2 & \text{if } x=2+3i \\ 4 & \text{if } x=3+3i \end{cases} \quad i=0,1,2,\ldots$$

To sum up, the recursive program $P_{1,2}$ has exactly three fixedpoints, each generated by a different selection of a solution to the above equations:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

$$g(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1 \\ 2 & \text{if } x=2 \\ 4 & \text{if } x=3 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

$$h(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1+3i \\ 2 & \text{if } x=2+3i \\ 4 & \text{if } x=3+3i \end{cases} \quad i=0,1,2,\ldots$$

Note that f is less defined than g and g is less defined than h . The only maximal fixedpoint of this program is h , and thus it is also the program's optimal fixedpoint.

Program $P_{1,3}$:

$$F(x) \Leftarrow \textit{if } x=0 \textit{ then } 1 \textit{ else } 3\cdot F(F(x-1)) .$$

As before, $F(0)=1$, and there are exactly two possible values for $F(1)$:

$$F(1) = \textit{undefined} \quad \text{or} \quad F(1) = 0 .$$

The first possibility leads to the same least fixedpoint as before:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The second possibility leads to:

$$F(2) = 3 \cdot F(F(1)) = 3 \cdot F(0) = 3 ,$$

$$F(3) = 3 \cdot F(3) .$$

Here we have the same choice once more,

$$F(3) = \textit{undefined} \quad \text{or} \quad F(3) = 0 .$$

If we choose $F(3)=\textit{undefined}$ we get the fixedpoint

$$g(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1 \\ 3 & \text{if } x=2 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

However, if we choose $F(3)=0$ we continue with

$$F(4) = 3 \cdot F(F(3)) = 3 \cdot F(0) = 3 ,$$

$$F(5) = 3 \cdot F(F(4)) = 3 \cdot F(3) = 0 ,$$

and so on, and we obtain the third possible fixedpoint:

$$h(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1+2i \\ 3 & \text{if } x=2+2i \end{cases} \quad i=0,1,2,\ldots$$

The optimal fixedpoint of $P_{1,3}$ is clearly $h(x)$.

Program $P_{1,4}$:

$$F(x) \Leftarrow \textit{if } x=0 \textit{ then } 1 \textit{ else } 4 \cdot F(F(x-1)) .$$

This program behaves entirely differently from the cases considered previously. For $x=0$, we still get $F(0)=1$. For $x=1$, we get:

$$F(1) = 4 \cdot F(F(0)) = 4 \cdot F(1) ,$$

and we have the same choice as before,

$$F(1) = \textit{undefined} \quad \text{or} \quad F(1) = 0 .$$

If we take $F(1)=0$, we continue with:

$$F(2) = 4 \cdot F(F(1)) = 4 \cdot F(0) = 4 ,$$

and therefore:

$$F(3) = 4 \cdot F(F(2)) = 4 \cdot F(4) = 16 \cdot F(F(3)) .$$

Here we encounter a new problem: We do get an equation for the value of $F(3)$, but $F(3)$ is contained in another occurrence of F on the righthand side of the equation. Since we do not know the global behavior of this function, we cannot simply solve this equation. However, based upon results in number theory, it can be shown that any fixedpoint of this program must be *undefined* for $x \geq 3$. Therefore, the program $P_{1,4}$ has exactly two fixedpoints:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

and

$$g(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1 \\ 4 & \text{if } x=2 \\ \textit{undefined} & \text{otherwise .} \end{cases}$$

Since f is less defined than g, f is the program's least fixedpoint and g is the program's optimal (and maximal) fixedpoint. In contrast to programs $P_{1,0}$, $P_{1,2}$ and $P_{1,3}$, the optimal fixedpoint is not a total function, even though it is still more defined than the least fixedpoint.

Finally, we consider

Program $P_{1,5}$:

$F(x) \Leftarrow$ *if* $x=0$ *then* 1 *else* $5 \cdot F(F(x-1))$.

For $x=0$ we clearly have $F(0)=1$. For $x=1$, we have, as usual, the choice between $F(1)=$*undefined* and $F(1)=0$. If we take the second possibility, we get $F(2)=5$. The difficulty arises when considering the possible values of $F(3)$:

$$F(3) = 5 \cdot F(F(2)) = 5 \cdot F(5) = 25 \cdot F(F(4)) = 25 \cdot F(5 \cdot F(F(3))) .$$

This equation is too difficult to be immediately solved.

Based upon considerations which are beyond the scope of this paper, we can find the following two fixedpoints of $P_{1,5}$:

$$g_1(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1+2i \\ 5 & \text{if } x=2+2i \end{cases} \quad i=0,1,2,\ldots$$

and

$$g_2(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1+3i \\ 5 & \text{if } x=2+3i \\ 25 & \text{if } x=3+3i \end{cases} \quad i=0,1,2,\ldots$$

The optimal fixedpoint must be less defined than both of these two total (and therefore maximal) fixedpoints, so it can be defined only at arguments of the form $x=1+6i$ and $x=2+6i$, for $i=0,1,2,\ldots$. However, the function thus obtained is not a fixedpoint of the program (e.g., try $x=7$). It can be shown that the only two fixedpoints of $P_{1,5}$ which are less defined than this function are:

$$f(x) = \begin{cases} 1 & \text{if } x=0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

and

$$h(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x=1 \\ 5 & \text{if } x=2 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The function $f(x)$ is clearly the program's least fixedpoint. The fixedpoint $h(x)$ is fxp-consistent, since all its values are uniquely determined by the equations.

Since the optimal fixedpoint must be either $f(x)$ or $h(x)$, and the more defined function $h(x)$ is fxp-consistent, $h(x)$ is the program's optimal fixedpoint. Note the similarity between the optimal fixedpoints of $P_{1,4}$ and $P_{1,5}$ - both are defined only for x=0 , x=1 and x=2 ; in $P_{1,4}$ this is due to the lack of possible fixedpoints, while in $P_{1,5}$ it is due to their multiplicity.

One could continue to check all programs $P_{1,j}$ with j greater than 5 . However, we believe that the preceding examples sufficiently illustrate the variety of possible cases in the new optimal fixedpoint approach. It is especially interesting to note that although the least fixedpoint of all programs $P_{1,j}$ is the same, the sets of all fixedpoints, as well as the optimal fixedpoints, of these programs differ widely. We summarize this situation in Fig. 3, where we exhibit the sets of fixedpoints of programs $P_{1,0}$ to $P_{1,5}$. The least fixedpoint of any such program is represented by the lowest dot, while the optimal fixedpoint is represented by the dot surrounded by a circle.

In the examples considered so far, various techniques were used to find the correct value of the optimal fixedpoint. Some of these techniques are easily mechanizable, while others require deep mathematical knowledge. Unlike the least fixedpoint of a recursive program, the optimal fixedpoint need not be a computable function. Thus there cannot be a "complete" computation rule which always computes the optimal fixedpoint, but we can still hope to find good computation techniques which are applicable to large subsets of commonly used programs. The examples discussed in this section give the flavor of a few such techniques.

V. Proof Techniques

In this section we illustrate several techniques for proving properties of optimal fixedpoints. We wish to show that optimal fixedpoint f of a given recursive program P has some property Q[f] without actually computing the fixedpoint. The property Q is a functional predicate, which may characterize the overall behavior of f . For example, Q[f] can state that f is a total function, or that f equals some given function g , or that f is monotonically increasing over some ordered domain, etc.

Generally speaking, there are three elements involved in the process of proving properties of fixedpoints: A function f , a domain D , and a desired property Q . Any one of these three elements can be used as the basis for induction.

The two classical methods for proving properties of least fixedpoints use induction on the function and on the domain. In the *computational induction* method (deBakker and Scott [1969]), one first proves the property Q over D for a very simple function f_0, and then successively treats better approximations f_i of f .

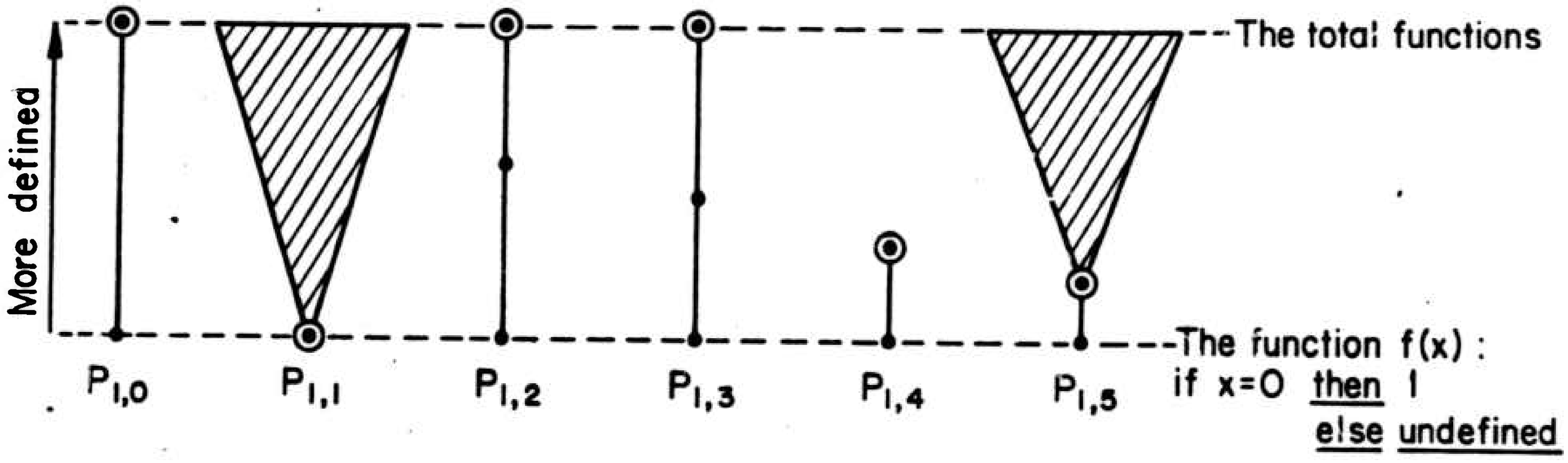

Fig. 3. The structure of the set of fixedpoints of the programs $P_{1,0}$ to $P_{1,5}$

In the *structural induction* method (Burstall [1969]) one uses induction over the elements of the domain D, leaving f and Q unchanged.

While these two general methods, appropriately modified, can also be used to prove properties of the optimal fixedpoint in some cases, we suggest a new induction method (called *assertion induction*) which uses the property Q as the basis for induction. Even though this third type of induction has been totally ignored in the least fixedpoint approach, it turns out to be a very useful technique in the optimal fixedpoint approach.

What we actually prove in the assertion induction method is that any fixedpoint f of the program belonging to some given subset S of partial functions has the property $Q[f]$. The fact that the optimal fixedpoint g possesses the desired property is derived either as a special case (if $g \in S$), or as a result of some further argumentation (based on the definition of g as the greatest fixedpoint which is fxp-consistent).

Note that S may contain functions which are not fixedpoints of the program, and these functions need not have the property Q . The assertion induction method only shows that all functions in S which are fixedpoints of the program have property Q . The role of the subset S is to rule out certain unwanted fixedpoints which do not have the desired property Q .

The Assertion Induction Method

Given: A recursive program $P : F(x) \Leftarrow \tau[F](x)$, a property $Q[F]$, and a subset S of partial functions.

Goal: To prove that $Q[f]$ holds for any fixedpoint f of P such that $f \in S$.

Method: Find a sequence of predicates $Q_i[F]$, $i=0,1,2,...$ such that:

(a) $Q_0[f]$ holds for any $f \in S$.

(b) If $Q_i[f]$ holds for some $f \in S$ and $\tau[f] \in S$, then $Q_{i+1}[\tau[f]]$ holds.

(c) For any $f \in S$, if $Q_i[f]$ holds for all i , then $Q[f]$ also holds.

This method can be justified by the following argument: By part (a), any fixedpoint $f \in S$ has property $Q_0[f]$. By part (b), if a function $f \in S$ has property

$Q_i[f]$, and $\tau[f]\epsilon S$, then $\tau[f]$ has property $Q_{i+1}[\tau[f]]$. But if f is a fixedpoint, then $f=\tau[f]$ so $\tau[f]\epsilon S$, and f has property $Q_{i+1}[f]$. By induction, any fixedpoint $f\epsilon S$ has the properties $Q_i[f]$ for $i=0,1,2,\ldots$. Thus, part (c) implies that f has property $Q[f]$. Note that since f is replaced by $\tau[f]$ in the induction step, any f which is not a fixedpoint of τ is not guaranteed to have all the properties Q_i .

We illustrate this method with the following recursive program over the natural numbers:

P4: F(x) <== *if* F(x+1)>0 *then* F(x+1)+1 *else* 0 .

The least fixedpoint of this program is everywhere *undefined*. We would like to prove that the optimal fixedpoint of this program is the constant function

f(x)=0 for any natural number x .

We first prove two properties of the fixedpoints of P4 which enable us to properly choose the subset S of partial functions:

(i) *For any fixedpoint f of P4 and for any natural number x ,*
f(x+1) is undefined if and only if f(x) is undefined.

To show this, assume that f(x+1) is *undefined*; then clearly $\tau[f](x)$ = *if* f(x+1)>0 *then* f(x+1) *else* 0 cannot be defined. Since $f(x)=\tau[f](x)$, f(x) is also *undefined*. On the other hand, if f(x+1) is defined, then $\tau[f](x)$ is also defined, and since $f(x)=[f](x)$, f(x) is defined.

(ii) *For any fixedpoint f of P4 and for any natural number x ,*
f(x+1)=0 if and only if f(x)=0 .

This can be shown in exactly the same way as in part (i) above.

These two properties characterize two possible fixedpoints of the program P4 : f which is everywhere *undefined* and g which is everywhere zero. Our aim now is to show that the recursive program has no other fixedpoints, and therefore while f is the program's least fixedpoint, g is the program's optimal fixedpoint.

The above two properties imply that any fixedpoint of P4 is either totally defined or totally *undefined*, and that for any total fixedpoint h , either h(x)=0 for all x or h(x)≠0 for all x . Therefore we define S as the set of all total functions which are everywhere greater than zero, and try to prove that P4 has no fixedpoint in S .

In order to achieve this, we formally define the predicate $Q[f]$ to be always "false". The sequence of intermediate predicates we use is:

$Q_i[f]$ is true if and only if $f(x)>i$ for all natural numbers x.

Step a: By the definition of S, any $f \in S$ is everywhere greater than zero, and therefore $Q_0[f]$ holds.

Step b: Suppose $Q_i[f]$ holds for some i and $f \in S$. Then by definition, $f(x)>i$ for all natural numbers x. Using this property, we can simplify the expression $\tau[f](x)$:

$$\begin{aligned}\tau[f](x) &= \textit{if } f(x+1)>0 \textit{ then } f(x+1)+1 \textit{ else } 0\\ &= f(x+1)+1.\end{aligned}$$

Since $f(x+1)>i$, we have $\tau[f](x)>i+1$. Therefore $Q_{i+1}[\tau[f]]$ also holds.

Step c: Suppose that some total function $f \in S$ satisfies $Q_i[f]$ for all i. Then for any natural number x, $f(x)>i$ for all i, and this is clearly a contradiction. Therefore any such f also satisfies $Q[f]$ which is always "false".

This completes the induction step, and the method thus guarantees that S does not contain any fixedpoint of P4.

Thus far we have introduced the new assertion induction method. As mentioned above, the two classical proof methods can also be used to prove properties of the optimal fixedpoint. We show here an appropriately modified version of the structural induction method.

The Structural Induction Method

The structural induction method is intended to prove that a fixedpoint f of a recursive program P has some "pointwise" property $Q[f](x)$ for all x in the domain D. The main idea is to partition D into subsets $S_0, S_1, \ldots$ such that

$$D = \bigcup_{j=o}^{\infty} S_j$$

and to prove that Q[f](x) holds for all $x \epsilon S_i$ using induction over the index i . Thus, one has to show that for any i , if Q[f](x) holds for all

$$x \epsilon \bigcup_{j=0}^{i-1} S_j$$

then Q[f](x) holds for all $x \epsilon S_i$.

This implication is usually proved by freely replacing any occurrence of f by τ[f] (since f is a fixedpoint) and applying the induction hypothsis to the resultant expression. This method can also be used to prove properties of optimal fixedpoints, but one usually has to apply some additional specific reasoning techniques, such as equation solving or case analysis of possible values.

We illustrate this method with the following program P5 over the natural numbers:

P5: F(x) <== *if* x=0 *then* 0 *else* F(x-F(x)) .

We would like to prove that the optimal fixedpoint f of P5 satisfies:

Q[f](x) : f(x)=0

for any natural number x .

We partition the domain of natural numbers in the following way:

S_0={0} S_1={1} S_2={2} ...

The fact that Q[f](0) holds (i.e., f(0)=0) is a direct consequence of the definition of P5 .

Assume that we have already shown that Q[f](x) holds for all $x \epsilon \bigcup_{j=0}^{i-1} S_j$

(i.e., for all $0 \le x \le i-1$); we now prove that Q[f](x) holds for all $x \epsilon S_i$ (i.e., for x=i). Since f is a fixedpoint of P5 and i>0 , we have:

f(i) = f(i-f(i)) .

We use case analysis in order to find all the possible values of f(i) .

One possible value of f(i) is clearly *undefined*. In order to check whether f(i) has any possible defined value, assume that $f(i)=k$ for some natural number k. Substituting this value into the definition of $f(i)$, we get:

$$k = f(i) = f(i-f(i)) = f(i-k) .$$

We consider two possible cases:

(a) If $k=0$, we obtain the requirement

$$0 = f(i)$$

and this value is clearly consistent with our assumption that $f(i)=k=0$. Thus zero is a possible value of $f(i)$.

(b) If $k>0$, we obtain the requirement that

$$f(i-k) > 0 ,$$

but since $i>0$ and $k>0$, $i-k<i$, and this contradicts what we know (in the induction hypothesis) about the optimal fixedpoint:

$$f(x)=0 \text{ for all } x ,\ 0 \le x \ge i .$$

Therefore $f(i)$ cannot have the value k for any $k>0$.

We have thus shown that the only two possible values of $f(i)$ are *undefined* and 0. By the definition of the optimal fixedpoint, we can now deduce that $f(i)=0$. Since this holds for any natural number i, the optimal fixedpoint is everywhere defined as zero.

VI. Conclusion

In this paper we have presented the optimal fixedpoint approach towards recursive programs. While it is clearly appealing from a theoretical point of view, it has a drawback in practice: it may be either impossible or extremely hard to find the optimal fixedpoint of some recursive programs. While we cannot develop perfect implementations, we can try (perhaps using heuristic techniques) to extract as much information from the program as possible. Such an implementation will yield the optimal fixedpoint for certain classes of recursive programs; it will compute some intermediate fxp-consistent fixedpoint for other classes; and in the worst case will yield the least fixedpoint of the program (as computed by the classical stack implementation). By insisting on finding a more informative solution of a recursive program than the least fixedpoint, it is natural that the efficiency of computation rules is reduced and the complexiity of proof techniquee is increased.

The development of this new approach is still underway, both in its theoretical and practical aspects.

****

Footnotes

(i) All functions in this program map natural numbers into natural numbers; thus, x-1 is defined to be 0 for x=0.

(ii) The theorem is proved in Manna and Shamir [1975], Theorem 3.

(iii) For a more rigorous statement of this result and its proof see Theorem 5 in Manna and Shamir [1975].

Acknowledgement. We are indebted to Nachum Dershowitz, Steve Ness and Richard Waldinger for their critical reading of the manuscript.

References

1. BURSTALL [1969].
   Burstall, R. M. Proving Properties of Programs by Structural Induction.
   Computer J., Vol. 12, No. 1 (Feb. 1969), pp. 41-48.

2. DeBAKKER and SCOTT [1969].
   DeBakker, J. W. and Scott, D. A Theory of Programs.
   Unpublished memo (Aug. 1969).

3. MANNA [1974].
   Manna, Z. *Mathematical Theory of Computation.*
   McGraw-Hill, N.Y. (1974).

4. MANNA and SHAMIR [1975].
   Manna, Z. and Shamir, A. The Optimal Fixedpoint of Recursive Programs.
   Proc. of the Symposium on Theory of Computing, Albuquerque, New Mexico (May 1975).

5. SHAMIR [1976].
   Shamir, A. The Fixedpoints of Recursive Programs.
   Ph.D. Thesis, Applied Mathematics Dept., Weizmann Institute of Science, Rehovot, Israel (1976).